NumPy Package

Python Modules and Packages: importing Modules

- **Standard Modules:**
  - Modules included with Python's standard library (e.g., math, datetime, os).
  - Example:

```
import math
print(math.sqrt(16)) # Output: 4.0
```

Third-Party Modules:

- Modules installed from external sources (e.g., requests, numpy, pandas).
- Example:

```
import requests
response = requests.get('https://api.example.com')
print(response.status_code)
```

Package installation

- What is pip?
  - The package installer for Python.
  - Used to install and manage third-party Python packages.
- Basic Commands:
  - Install a package: pip install package_name
  - Upgrade a package: pip install --upgrade package_name
  - Uninstall a package: pip uninstall package_name
  - List installed packages: pip list

Package installation

- What is conda?
  - An open-source package management system and environment management system.
  - Handles package installations, dependencies, and environments.
- Key Features:
  - Manage environments: Create, export, list, remove environments.
  - Install packages: conda install package_name
  - Create and activate environments: conda create --name myenv and conda activate myenv

Installation of numpy using pip OR conda

- Using pip

  pip install numpy

- Using conda

  conda install numpy

How to import the package

- To access NumPy and its functions import it in your Python code like this:

```
import numpy as np
```

- We shorten the imported name to np for better readability of code using NumPy.

NumPy

- NumPy (Numerical Python) is a fundamental package for scientific computing in Python.
- Provides support for arrays, matrices, and a large collection of mathematical functions to operate on these data structures.
- Foundation for other scientific libraries such as SciPy, Pandas, and scikit-learn.

Creating Arrays

- Creating a NumPy ndarray object
  - 1D array

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

```
[1 2 3 4 5]
```

Numpy Array vs. Python List?

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(type(arr))
```

```
<class 'numpy.ndarray'>
```

```
lists=[1,2,3,4]
print(type(lists))
```

```
<class 'list'>
```

Numpy Array vs. Python List?

Python List	Numpy Array
store elements of different types in a single list (e.g., integers, strings, floats). Example: [1, 'text', 3.14]	stores elements of the same type. Example: np.array([1, 2, 3])
Not optimized formatrix computation	highly optimized for performance, especially for numerical operations: contiguous memory allocation and vectorized operations, making it faster for large-scale mathematical computations.
Supports basic list operations such as appending, slicing, and iteration. Mathematical operations are not supported directly; you need to use loops or list comprehensions.	Supports a wide range of mathematical operations, including element-wise operations, matrix operations, and advanced mathematical functions. Includes a rich set of functions for operations like linear algebra, statistics, and Fourier transforms.
Does not support broadcasting, Operations between lists of different sizes require manual handling or looping.	Supports broadcasting, allowing operations between arrays of different shapes in a way that automatically aligns the shapes.

NumPy array using Tuple

```
import numpy as np
arr = np.array((1, 2, 3, 4, 5))
print(arr)
```

```
[1 2 3 4 5]
```

Access Array Elements

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

```
1
```

Access Array Elements

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[2] + arr[3])
```

```
7
```

Access 2D Array Elements

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr)
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
```

How to access 5th element of 2nd row?

Access 2D Array Elements

How to access 4th element of 2nd row?

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr[1, 3])
```

```
9
```

Try some example:

- Access 2nd element of 1st row.
- Access 5th element of 2nd row.

Access 3D Array Elements

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr)
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

Access 3D Array Elements

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

```
6
```

Negative Indexing

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(arr[1, -1])
```

```
10
```

NumPy Array Slicing

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

```
[2 3 4 5]
```

NumPy Array Slicing

Slice elements from index 4 to the end of the array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

```
[2 3 4 5]
```

We pass slice instead of index like this: [start:end].

We can also define the step, like this: [start:end:step]

NumPy Array Slicing

Slice elements from index 4 to the end of the array and Slice elements from the beginning to index 4 (not included):

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
print(arr[:4])
```

```
[5 6 7]
[1 2 3 4]
```

If we don't pass start its considered 0

If we don't pass end its considered length of array in that dimension

If we don't pass step its considered 1

NumPy Array Negative Slicing

Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])
```

```
[5 6]
```

NumPy Array Slicing with Step

Return every other element from index 1 to index 5:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
```

```
[2 4]
```

Try some example:

- Return every other element from the entire array:

NumPy 2D Array Slicing with Step

Return every other element from index 1 to index 5:

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

```
[7 8 9]
```

Data Type of an Numpy Array

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

```
int64
```

Data Type of an Numpy Array

Below is a list of all data types in NumPy and the characters used to represent them.

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type (void)

Copy Numpy Array

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[1 2 3 4 5]
```

View Numpy Array

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42
print(arr)
print(x)
```

```
[42  2  3  4  5]
[42  2  3  4  5]
```

Shape Numpy Array

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

```
(2, 4)
```

Reshape Numpy Array

Convert 1-D array with 12 elements into a 2-D array.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Try some example:

- Convert 1-D array with 12 elements into a 3-D array.

Flattening the arrays

Flattening array means converting a multidimensional array into a 1D array.

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
newarr = arr.reshape(-1)
print(newarr)
```

```
[1 2 3 4 5 6]
```

Joining NumPy Arrays

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))
print(arr)
```

```
[1 2 3 4 5 6]
```

Broadcasting

- Broadcasting allows NumPy to perform arithmetic operations on arrays of different shapes.

  - Suppose, we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array.
  - For example, suppose that we want to add a constant vector to each row of a matrix.